REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
MARCH 2009	Conference Paper Preprint	March 2008 – March 2009

4. TITLE AND SUBTITLE

CELLULAR CLASS ENCODING APPROACH TO INCREASING EFFICIENCY OF NEAREST NEIGHBOR SEARCHING (PREPRINT)

5a. CONTRACT NUMBER
FA8750-08-C-0029

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
35885G

6. AUTHOR(S)

Mark Huggins, Aaron Lawson, and Brett Smolenski

5d. PROJECT NUMBER
3188

5e. TASK NUMBER
TW

5f. WORK UNIT NUMBER
EN

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Oasis Systems, Inc.
81 Hartwell Avenue
Lexington, MA 02421

RADC, Inc.
10002 Hillside Terrace
Marcy, NY 13403

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RIEC
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
N/A

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TP-2010-9

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited. PA # 88ABW-2009-1235 Date Cleared: 26-March-2009

13. SUPPLEMENTARY NOTES
This work, resulting in whole or in part from Department of the Air Force contract number FA8750-08-C-0029, has been submitted to and accepted for publication in the March 2010 IEEE International Conference on Acoustics, Speech and Signal Processing proceedings. If this work is published, IEEE may assert copyright. The United States has for itself and others acting on its behalf an unlimited, paid-up, nonexclusive, irrevocable worldwide license to use, modify, reproduce, release, perform, display, or disclose the work by or on behalf of the Government. All other rights are reserved by the copyright owner.

14. ABSTRACT
Nearest neighbor searching (NNS) is a common classification method, but its brute force implementation is inefficient for dimensions greater than 10. We present Cellular Class Encoding (CCE) as an alternative, full-search equivalent shown to be 1.1-1.8 times faster than BF on real-world, 14-dimensional data sets. Given a query in an indexed cell of a partitioned space, the CCE's efficiency is achieved by only performing NNS on those database elements which could not be eliminated *a priori* as impossible nearest neighbors of vectors residing in that cell. To ensure CCE is a viable alternative for real-world applications, we use VQ Speaker ID as a testbed application and present results.

15. SUBJECT TERMS

Nearest neighbor search, vector quantization, speaker identification

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			John G. Parker
U	U	U	UU	5	19b. TELEPHONE NUMBER (*Include area code*) N/A

CELLULAR CLASS ENCODING APPROACH TO INCREASING EFFICIENCY OF NEAREST NEIGHBOR SEARCHING

Mark Huggins[1], Aaron Lawson[2], Brett Smolenski[2]

[1] Oasis Systems, Inc., [2] RADC, Inc.

ABSTRACT

Nearest neighbor searching (NNS) is a common classification method, but its brute-force (BF) implementation is inefficient for dimensions greater than 10. We present Cellular Class Encoding (CCE), shown to be 1.1-1.7 times faster than BF on real-world, 14-dimensional data sets. Moreover, if applied to bounded sets, CCE is a full-search equivalent to BF.

Given a query in an indexed cell of a partitioned bounded space, the CCE's efficiency is achieved by only performing NNS on those database elements which could not be eliminated a priori as impossible nearest neighbors of that cell's resident vectors. To ensure CCE is a viable alternative in real-world applications, we use VQ speaker identification as a testbed application and present results.

Index Terms— Nearest neighbor search, vector quantization, cellular class encoding, vector approximation-file

1. INTRODUCTION

Nearest neighbor searching (NNS) is a common classification method in pattern recognition systems such as Vector Quantization (VQ). The main drawback with this approach is that the full-search (a.k.a. brute-force) technique is very inefficient for dimensions greater than 10 which is only compounded when large database sizes are involved. Alternative full-search equivalents (FSE) to BF do exist—one of the more successful of these reported is the Approximating and Eliminating Search Algorithm (AESA) [1] and its derivatives such as Linear AESA [2] and Reduced Overhead AESA [3]. Tests of these three alternatives failed to be any faster than BF for our testbed application of VQ speaker ID.

The Vector Approximation-File (VA-F) method presented in [4] is a viable FSE alternative in real-world applications. Our implementation of this also failed to be any faster than BF; however, there were some useful ideas in VA-F that we felt could be modified for our purposes—thus inspiring the Cellular Class Encoding (CCE) method.

Given any test vector in an indexed cell of a partitioned space, the CCE's efficiency is achieved by only performing NNS on those database elements which could not be eliminated a priori as impossible nearest neighbors of any vector residing in that cell. It is the indices of the non-eliminated database elements which form the class for that cell. To ensure CCE was a viable alternative in real-world applications, we used VQ speaker identification (SID) as a testbed application—results are reported on population ranging from 10 to 1500 enrolled speakers.

In addition to its use in SID, VQ is an essential component of many audio compression techniques. For example, CELP and systems that use CELP such as SPEEX, G.729, TwinVQ, Vorbis, AMR-WB+, and DTS all rely on VQ codebook searches to enable their lossy compression techniques. Hence, a full-search equivalent that is more efficient than BF for dimensions greater than 10 will find broader applicability than just SID.

2. VECTOR APPROXIMATION–FILE DESCRIPTION

Henceforth, we shall use the following notation: $\vec{v}_q$ to denote a query vector, $\vec{v}_i$ for the *i*-th vector of the database, $v_{i,j}$ for the *j*-th component of $\vec{v}_i$, b_j for the number of bits for approximating values in the *j*-th dimension, $p_j[k]$ for the *k*-th partition point in the *j*-th dimension, and $r_{i,j}$ for the region in which $\vec{v}_i$ resides in the *j*-th dimension or alternatively $r_{i,j}[n]$ shall mean $r_{i,j} = n$.

In the VA-F method, *b* bits are allotted to partition a bounded region of a *d*-dimensional vector space into 2^b uniform cells. Each cell is uniquely bit-encoded which serves to quantize its resident vectors. To aid our description, we summarize the example given in [4]. In this example, 3 bits encode the partitioning of a bounded plane region such that 2 bits encode the *x*-dimension's partitioning into 4 regions, 1 bit encodes the partitioning of the *y*-dimension into 2 regions (see Figure 1). For a generic query vector $\vec{v}_q = (v_{q,0}, v_{q,1})$, the partition points $p_0[2]$ and $p_0[3]$ which bound region $r_{q,0}[2]$ are such that $p_0[2] \le v_{q,0} < p_0[3]$. Thus, we say component $v_{q,0}$ resides in region $r_{q,0}[2]$ and encode component $v_{q,0}$'s approximation as the binary equivalent of this region's decimal index 2. That is, we encode $v_{q,0}$'s approximation as the bit-string '10'. Likewise, the partition points $p_1[1]$ and $p_1[2]$ which bound $r_{q,1}[1]$ are such that $p_1[1] \le v_{q,1} < p_1[2]$. Hence, component $v_{q,1}$ resides in region $r_{q,1}[1]$ and we encode its approximation as the bit-string '1'. Thus, the bit-encoded vector approximation for $\vec{v}_q$ is just the concatenation of its resident regions' bit-encodings, yielding the bit-string '101'. Finally, the VA-File for a database of *N* vectors is one long array of all *N* vectors' bit-encoded approximations, plus supporting information such as the number of bits, b_j, per approximation in the *j*-th dimension and the partition points $p_j[0], \cdots, p_j[2^{b_j}]$ that define the *j*-th dimension's partition.

In subsequent database searches for the *k* nearest-neighbors of a query vector $\vec{v}_q$, we sequentially scan each database element $\vec{v}_i$'s approximation in the VA-file, while simultaneously computing a lower bound l_i and upper bound u_i for the L_p-distance $d_{Lp}(\vec{v}_q, \vec{v}_i)$. These bounds are computed as

$$l_i = \left(\sum_{j=0}^{d-1} \left(l_{i,j} \right)^p \right)^{\frac{1}{p}} \tag{1}$$

where

$$l_{i,j} = \begin{cases} v_{q,j} - p_j\left[r_{i,j}+1\right] & r_{i,j} < r_{q,j} \\ 0 & r_{i,j} = r_{q,j} \\ p_j\left[r_{i,j}\right] - v_{q,j} & r_{i,j} > r_{q,j} \end{cases} \tag{2}$$

and

$$u_i = \left(\sum_{j=0}^{d-1} \left(u_{i,j} \right)^p \right)^{\frac{1}{p}} \tag{3}$$

where

$$u_{i,j} = \begin{cases} v_{q,j} - p_j\left[r_{i,j}\right] & r_{i,j} < r_{q,j} \\ \max\left(v_{q,j} - p_j\left[r_{i,j}\right], p_j\left[r_{i,j}+1\right] - v_{q,j} \right) & r_{i,j} = r_{q,j} \\ p_j\left[r_{i,j}+1\right] - v_{q,j} & r_{i,j} > r_{q,j} \end{cases} \tag{4}$$

Hence, a database element $\vec{v}_i$ is a nearest neighbor (NN) candidate if less than k vectors have been scanned thus far or if its lower

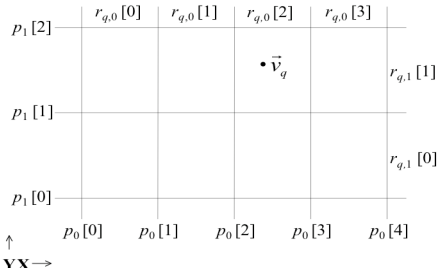

Fig. 1. Example of using a 3-bit allocation to encode the partitioning of the xy-plane.

bound l_i is less than the k-th element in the current candidate set. Once all vector approximations in the database have been scanned, the actual L_p-distance is computed only on those elements of the final candidate set. It is this filtering out of database elements which gives the boost in efficiency in this NNS approach.

3. CELLULAR CLASS ENCODING DESCRIPTION

As a candidate alternative to the brute-force (BF) method of NNS, we were hopeful when we applied the VA-F to the real-world task of VQ SID. We were surprised and disappointed when BF outperformed the VA-F method. It is unclear why this is so, but we surmise that there are some less understood correlations in real-world data of this type which reduce the VA-F's filtering capacity during the NNS. These disappointing results motivated us to develop the Cellular Class Encoding (CCE) method of NNS.

3.1. Overview

The CCE method is an extension of VA-F in that we borrow from it the techniques of space partitioning, bit-encoded approximations and computation of the lower/upper distance bounds. What differentiates CCE from VA-F is the distance bound computations are only performed offline and not at query time. Additionally, in the offline stage the distance bounds are used to map a cell of the partitioned space to a subset of database elements that cannot be eliminated as impossible NNs of any vector residing in that cell. Consequently, during future NN queries we avoid the lower/upper bound computations required to find a query vector's NN candidates. Instead, we only need to determine the query vector's resident cell, from which we directly proceed to computing the actual L_p-distance between the query vector and that cell's set of NN candidates.

3.2. Encoding steps

To show how to encode a cell's NN candidate class, suppose our database is composed of N, d-dimensional vectors from a bounded feature space which we partition into 2^b uniform cells. Hence, b bits can be used to encode this partitioning. Note that each cell C_k will be a d-dimensional hypercube; let us use $r_{k,j}$ to denote the region in the j-th dimension intersected by C_k. Thus, $r_{k,j}$ is bounded by partition points $p_j\left[r_{k,j}\right]$ and $p_j\left[r_{k,j}+1\right]$. Hence, for each vector $\vec{v} \in C_k$ and each database element $\vec{v}_i$ we compute the lower and upper bounds l_i and u_i of the distance $d_{Lp}\left(\vec{v},\vec{v}_i\right)$ as

$$l_i = \left(\sum_{j=0}^{d-1} \left(l_{i,j} \right)^p \right)^{\frac{1}{p}} \tag{5}$$

where

$$l_{i,j} = \begin{cases} p_j\left[r_{k,j}\right] - v_{i,j} & r_{i,j} < r_{k,j} \\ 0 & r_{i,j} = r_{k,j} \\ v_{i,j} - p_j\left[r_{k,j}+1\right] & r_{i,j} > r_{k,j} \end{cases} \tag{6}$$

and

$$u_i = \left(\sum_{j=0}^{d-1} \left(u_{i,j} \right)^p \right)^{\frac{1}{p}} \tag{7}$$

where

$$u_{i,j} = \begin{cases} p_j\left[r_{k,j}+1\right] - v_{i,j} & r_{i,j} < r_{k,j} \\ \max\left(v_{i,j} - p_j\left[r_{k,j}\right], p_j\left[r_{k,j}+1\right] - v_{i,j} \right) & r_{i,j} = r_{k,j} \\ v_{i,j} - p_j\left[r_{k,j}\right] & r_{i,j} > r_{k,j} \end{cases} \tag{8}$$

Let us denote the least upper bound of the database elements with respect to cell C_k as

$$\text{lub}_k = \min_{i=0}^{N-1}\{ u_i \} \tag{9}$$

Hence, a database element $\vec{v}_i$ can be eliminated as a NN of all vectors in cell C_k if its lower bound l_i is greater than the smallest upper bound lub_k. Hence, the indices of non-eliminated database elements comprise the cellular class, $\hat{C}_k$, of cell C_k. That is,

$$\hat{C}_k = \{ i \mid l_i \le \text{lub}_k \} \tag{10}$$

Moreover, if for example, database elements 8, 12 and 19 make up this class, we compactly encode that by setting 8th, 12th and 19th bits of an unsigned, N-bit integer to '1' and all others set to '0'. So in addition to the usual information saved for VA-F, in CCE method we also save the N-bit encodings for all cellular classes.

3.2. Decoding steps

To find the closest database element of a query vector $\vec{v}_q$, the first step is to determine the cell, C_q, of the previously partitioned feature space that $\vec{v}_q$ resides—that of course is the one indexed by

$\vec{v}_q$'s bit-encoded approximation (refer to Section 2 on how this is done). Secondly, having C_q allows us direct access to its class $\hat{C}_q$—its candidate NNs' indices. Lastly, the closest database element of $\vec{v}_q$ is computed as

$$NN(\vec{v}_q) = \arg\min \{ \quad d_{Lp}(\vec{v}_q, \vec{v}_i) \mid i \in \hat{C}_q \quad \} \tag{11}$$

4. VQ CLASSIFIER FOR SPEAKER IDENTIFICATION

To ensure CCE was a viable alternative in real-world applications, VQ speaker identification (SID) was used as a testbed application. In SID, the database is the enrolled speaker models and a query is a feature vector from an unknown speaker we wish to match with an enrolled speaker.

Designing an optimal classifier for SID is a challenging task, since one would like to compare feature vectors originating from unique acoustic classes, such as, quasi-periodic, noise-like, and impulse-like speech sounds. The VQ approach obtains distinct acoustic classes without actually associating these classes to specific phonemes or other speech categories. This reduces the variation in the features due to phonetic differences while preserving the variation due to speaker differences [5]. However, being able to accurately detect these distinct acoustic states from the speech signal remains an active research area.

VQ uses the *k*-nearest neighbor clustering algorithm to find *k* unique cluster centroids, known as codebooks, which are derived from each speaker in the training data set. To recognize an unknown speaker, each test feature vector is compared to each enrolled speaker's codebook and the minimum distance is determined. The enrolled speaker with the smallest average of the above minimum distances is used to classify the test speaker as that enrolled member. For clustering and comparisons, we use the familiar Euclidean distance, also known as the L_2-distance.

5. EXPERIMENT SETUP

In this section we describe the hardware and operating system setup, data setup and the parameters for both the VQ and CCE implementation used in our testbed experiment.

5.1. Hardware/OS setup

Experiments were run using a 4 x 3.0 GHz AMD Opteron 8222 processor with 32 GB of RAM. Operating system was OpenSuSE 10.2 (64-bit), but the Speaker ID application was single-threaded and compiled in 32-bit mode.

5.2. Data description

We used a pool of 2185 speakers supplied from 3 Corpora: TIMIT, 2000 NIST SRE and an in-house corpus. The in-house corpus was conversational with the conversations held face-to-face with the microphone positioned on the table facing one speaker. The data collected from this primary speaker was the only data used in these experiments and the secondary speaker's audio was removed from all files. Recordings were done with a Samson C01U Condenser USB microphone in a low noise meeting room environment. A total of 632 speakers were collected with an average of about 100 seconds of combined audio per speaker.

From our speaker pool, we investigated subsets of 10, 20, …, 90, 300, 350, …, 600 and 1500 speakers. For each speaker set size, 30 subsets were randomly picked. For instance, we created 30 different subsets of 10 speakers, 30 different subsets of 20 speakers and so on. The statistics for the training and testing durations are summarized in Table 1. Note that there was no overlap between training and testing data. For each subset a single identification session was conducted, during which the average ID time per query vector was tracked. The timing result then for speaker sets of size *N* was the pooled average query time over its 30 subsets.

Table 1. Training and Testing Statistics

Training Duration (secs)				Testing Duration (secs)			
min	avg	std	max	min	avg	std	max
9	69	45	165	1	6	3	10

5.3. VQ implementation

Each speaker's model was composed of three separate 64-word codebooks: Linear Predictive Cepstra Coefficients (LPCC), LPCC-deltas, and Hamming liftered LPCC. The LPCCs were 14th order and computed over pre-emphasized, Hamming windowed, 32 ms frames at a 16 ms frame rate. An enrolled speaker's score against a test speaker's collection of query vectors was the equi-weighted sum of the 3 averaged feature scores (after each had been scaled by the minimum averaged feature score across speakers).

5.4. CCE implementation

For each 14-dimensional feature space, we allocated 16 bits for partitioning such that the 0th and 1st dimensions were allotted 2 bits each, and the remaining dimensions each were allotted 1 bit. This partitioning was speaker-specific—the following example explains what we mean by this: For the *j*-th dimension, speaker *i*'s LPCC codebook determined the partitioning of the bounded interval $[\lambda_{i,j}, \upsilon_{i,j}]$, such that $\lambda_{i,j}$ was equal to the mean, $\mu_{i,j}$, of the *j*-th components of speaker *i*'s LPCC codewords minus 2-standard deviations, $2 \cdot \sigma_{i,j}$. Similarly, $\upsilon_{i,j}$ was equal to $\mu_{i,j} + 2 \cdot \sigma_{i,j}$. So the partitioning points for $[\lambda_{i,j}, \upsilon_{i,j}]$ with respect to speaker *i* were

$$\begin{aligned} p_{i,j}[0] &= \lambda_{i,j} + 0 \cdot \delta, \\ p_{i,j}[1] &= \lambda_{i,j} + 1 \cdot \delta, \\ &\vdots \\ p_{i,j}[b_j^2] &= \lambda_{i,j} + b_j^2 \cdot \delta \end{aligned} \tag{12}$$

and where

$$\delta = \frac{\upsilon_{i,j} - \lambda_{i,j}}{b_j^2} \tag{13}$$

The delta and lifter codebooks' CCEs were similarly derived.

Although we originally intended CCE to be a full-search equivalent NNS approach, our implementation was not since the interval $[\lambda_{i,j}, \upsilon_{i,j}]$ did not guarantee to encapsulate all possible values that a query vector could take. However, this was not detrimental, as our results will demonstrate. Furthermore, the partitioning defined by (12) and (13) required we modify the region-assignment criteria for query vector $\vec{v}_q$'s components that fell outside of $[\lambda_{i,j}, \upsilon_{i,j}]$. In particular, if $v_{q,j} < \lambda_{i,j}$, then $v_{q,j}$ was assigned to region $r_{q,j}[0]$. Alternatively if $\upsilon_{i,j} < v_{q,j}$, then $v_{q,j}$ was assigned to region $r_{q,j}[2^{b_j} - 1]$. This assignment modification

raises two issues: 1) What if $\vec{v}_q$ is a representative vector from enrolled speaker *i*?, or 2) What if it is not? If the first case is true and if $v_{q,j} \notin [\lambda_{i,j}, \upsilon_{i,j}]$, then the *j*-th component $v_{q,j}$ is an outlier of enrolled speaker *i*, and approximating its value as one of the endpoint regions serves to correct its outlier status. If the second case is true, then the cellular class of enrolled speaker *i* to which $\vec{v}_q$ is mapped may refer to some candidate NNs from speaker *i* that are false candidates. Those false NNs will be eliminated during the actual L_2-distance calculations; the only harm done is the extra distance calculations. The timing results will bear out if these extra steps negate CCE as a viable alternative to BF. A 64-bit integer was used to encode each cell's class of candidate NN codewords.

6. EXPERIMENT RESULTS

6.1. Timing results

Figures 2 and 3 give the pooled average timing results for the examined speaker set sizes. In all cases, CCE outperformed BF—ranging from 1.6 to 1.7 times faster for less than 100 enrolled speakers down to 1.1 times faster for 1500 speakers (see Figure 4).

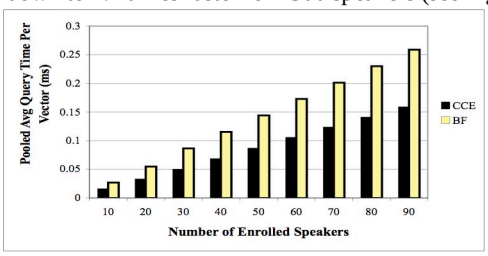

Fig. 2. CCE's vs. BF's pooled avg SID query time per test vector for 10 to 90 enrolled speakers.

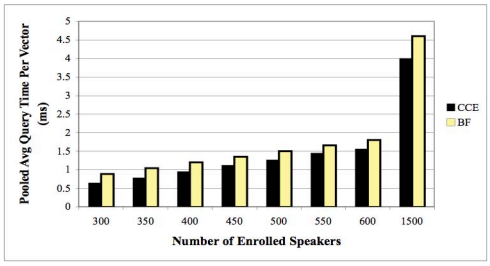

Fig. 3. CCE's vs. BF's pooled avg SID query time per test vector for 300 to 1500 enrolled speakers.

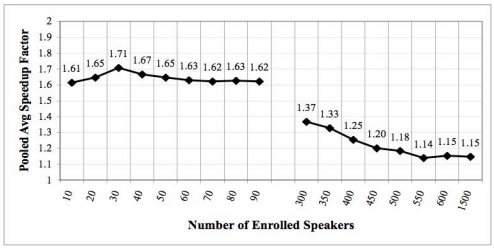

Fig. 4. CCE's pooled avg speedup factor over brute force.

6.2. Accuracy results

Since our CCE implementation was not a full-search equivalent to BF, it was important to ensure identification accuracies were comparable to BF. CCE does indeed maintain comparable accuracy, with some instances performing negligibly better or worse than BF (fig. 5).

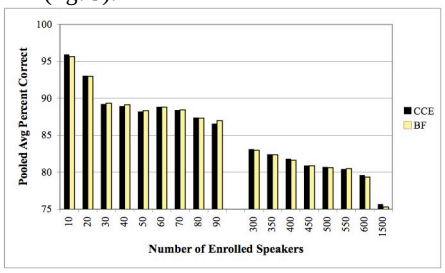

Fig. 5. CCE's vs. BF's avg pooled SID accuracy.

6.3. Memory requirements

Our CCE implementation required an additional 1,573,416 bytes per enrolled speaker—broken down as: (2^{16} cells) • (8 bytes / class) • (3 feature codebooks) to encode the cellular classes, plus (46 partition points across all 14 dimensions) • (4 bytes / float) • (3 feature codebooks) for the partition points.

7. DISCUSSION/CONCLUSIONS

The CCE search methodology for NNS has been demonstrated to be significantly more efficient than brute-force search in the context of VQ SID, and in preliminary benchmarks faster than both AESA and Vector Approximation-File. This increase in efficiency is especially robust for speaker sets of less than 100, where processing time was routinely reduced by 38 to 41%. Since larger speaker sets may always be broken down into smaller sets one can easily implement the CCE approach in a manner that reliably increases performance by 1.6 to 1.7 times faster. Furthermore, the CCE approach was able to maintain virtually identical speaker recognition accuracy when compared to BF.

Lastly, future research shall apply the CCE method to VQ optimization in the speech coding domain and real-time search applications. Whether non-uniform feature space partitioning can increase the CCE's filtering capacity will also be investigated.

11. REFERENCES

[1] Vidal Ruiz, E., "An algorithm for finding nearest neighbours in (approximately) constant average time", Pattern Recognition Letters, vol. 4, 1986.

[2] Mico, L., Oncina, J., Vidal, E., "An algorithm for finding nearest neighbours in constant average time with a linear space complexity", Pattern Recognition, Vol. II, 1992.

[3] Vilar, Juan Miguel, "Reducing the overhead of the AESA metric-space nearest neighbor searching algorithm", Information Processing Letters, Vol. 56, Issue 5, 8 December 1995.

[4] Blott, S. and Weber, R., "A simple vector-approximation file for similarity search in high-dimensional vector spaces", Technical Report 19, European ESPRIT project HERMES (project no. 9141), March 1997.

[5] Quatieri, T. F., Discrete-time Speech Signal Processing: Principles and Practice, Upper Saddle River, NJ: Prentice-Hall, 2002.